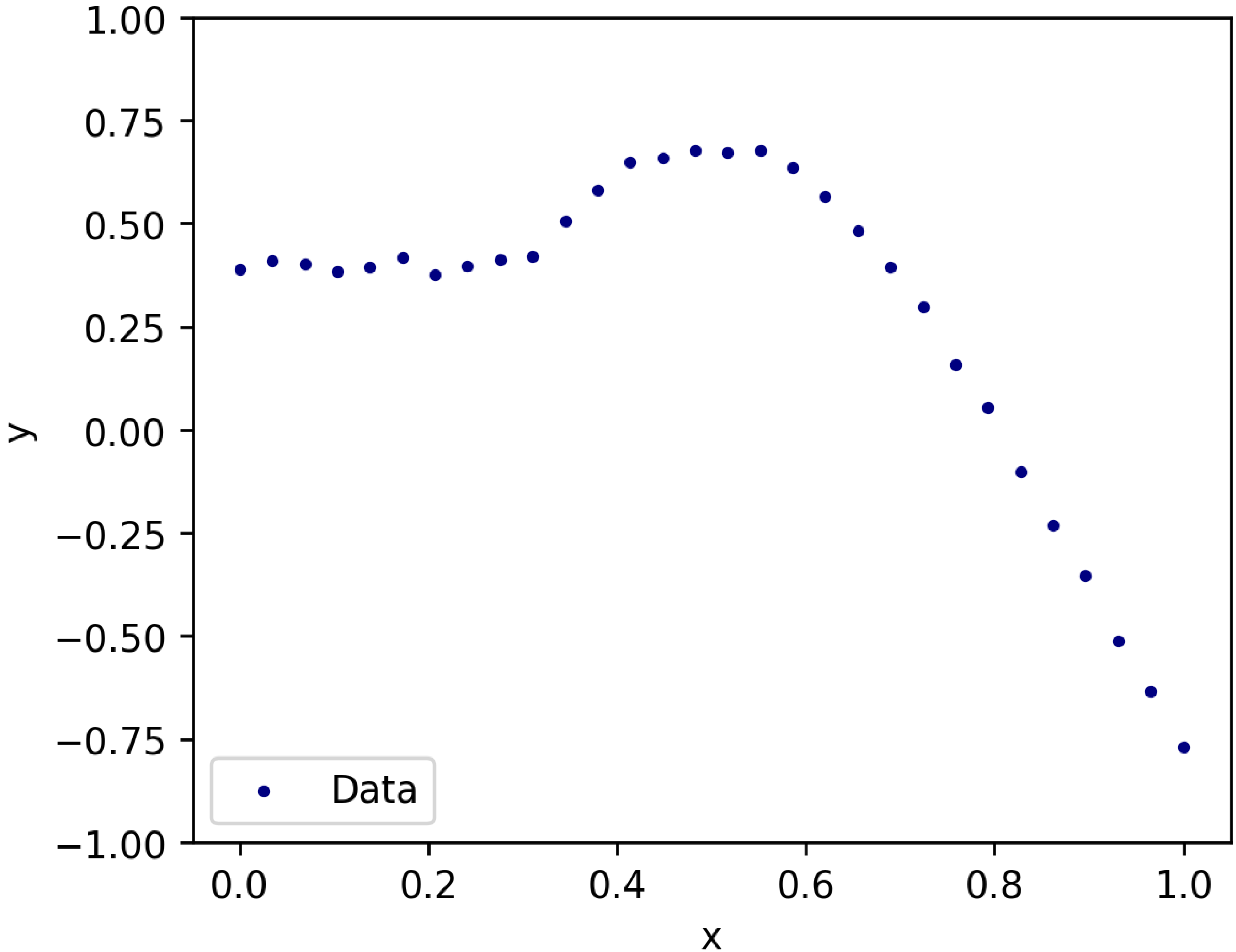
M7-L2 Problem 2

Here you will create a simple neural network for regression in PyTorch. PyTorch will give you a lot more control and flexibility for neural networks than SciKit-Learn, but there are some extra steps to learn.

Run the following cell to load our 1-D dataset:

In [34]:
```
import numpy as np
import matplotlib.pyplot as plt
import torch
from torch import optim, nn
import torch.nn.functional as F

x = np.array([0.        , 0.03448276, 0.06896552, 0.10344828, 0.13793103,0.17241379, 0.20689655, 0.24137931, 0.27586207
y = np.array([ 0.38914369,  0.40997345,  0.40282978,  0.38493705,  0.394214  ,0.41651437,  0.37573321,  0.39571087,  0.4


plt.figure(figsize=(5,4),dpi=250)
plt.scatter(x,y,s=5,c="navy",label="Data")
plt.legend(loc="lower left")
plt.ylim(-1,1)
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```



PyTorch Tensors

PyTorch models only work with PyTorch Tensors, so we need to convert our dataset into a tensors.

To convert these back to numpy arrays we can use:

- `x.detach().numpy()`
- `y.detach().numpy()`

In [35]:
```
x = torch.Tensor(x)
y = torch.Tensor(y)
```

PyTorch Module

We create a subclass whose superclass is `nn.Module`, a basic predictive model, and we must define 2 methods.

`nn.Module` subclass:

- `__init__()`
  - runs when creating a new model instance
  - includes the line `super().__init__()` to inherit parent methods from `nn.Module`
  - sets up all necessary model components/parameters
- `forward()`
  - runs when calling a model instance
  - performs a forward pass through the network given an input tensor.

This class `Net_2_layer` is an MLP for regression with 2 layers. At initialization, the user inputs the number of hidden neurons per layer, the number of inputs and outputs, and the activation function.

In [36]:
```
class Net_2_layer(nn.Module):
    def __init__(self, N_hidden=6, N_in=1, N_out=1, activation = F.relu):
        super().__init__()
        # Linear transformations -- these have weights and biases as trainable parameters,
        # so we must create them here.
        self.lin1 = nn.Linear(N_in, N_hidden)
        self.lin2 = nn.Linear(N_hidden, N_hidden)
        self.lin3 = nn.Linear(N_hidden, N_out)
        self.act = activation
```

```
    def forward(self,x):
        x = self.lin1(x)
        x = self.act(x)  # Activation of first hidden layer
        x = self.lin2(x)
        x = self.act(x)  # Activation at second hidden layer
        x = self.lin3(x) # (No activation at last layer)

        return x
```

Instantiate a model

This model has 6 neurons at each hidden layer, and it uses ReLU activation.

In [37]:
```
model = Net_2_layer(N_hidden = 6, activation = F.relu)
loss_curve = []
```

Training a model

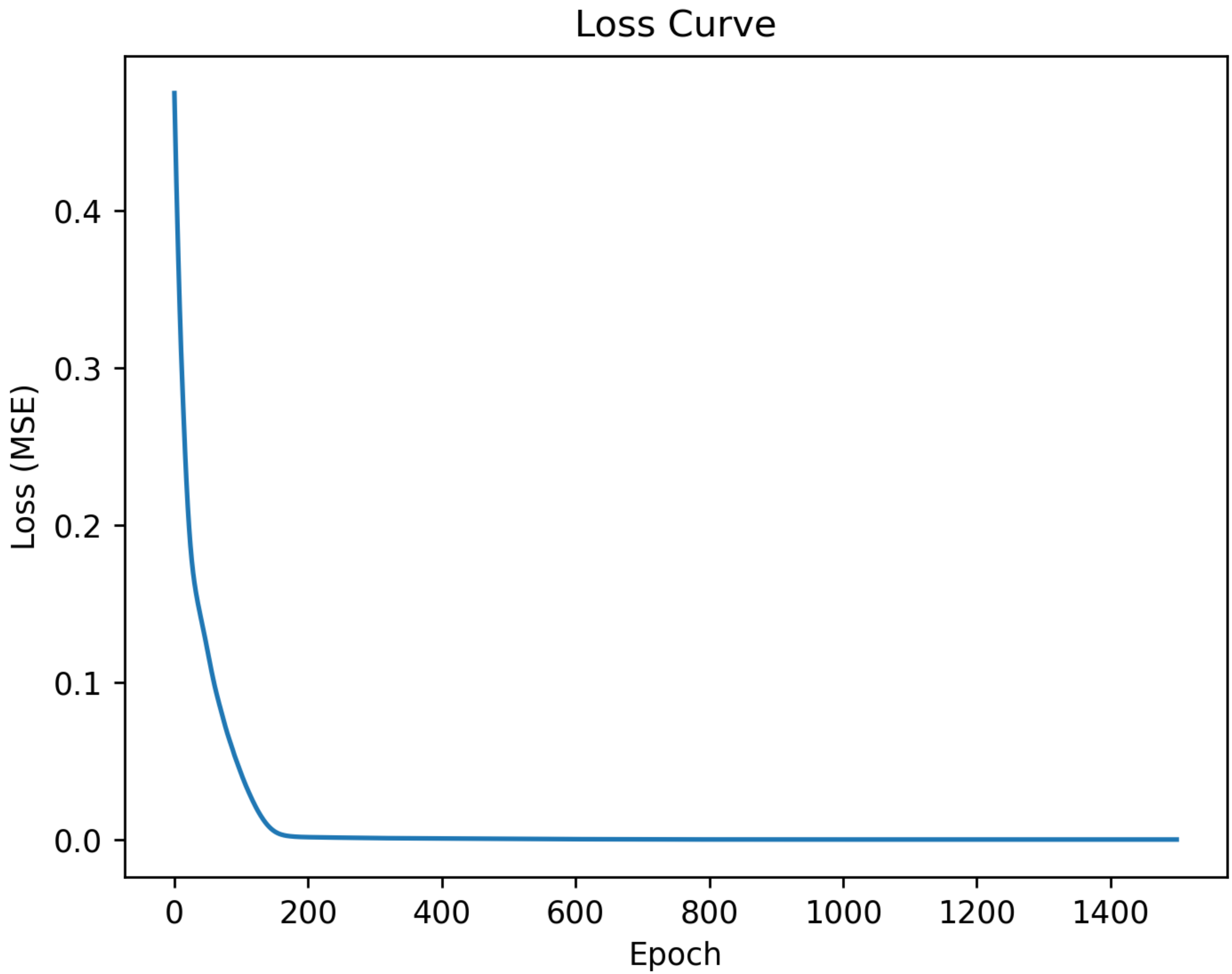
In [38]:
```
# Training parameters: Learning rate, number of epochs, loss function
# (These can be tuned)
lr = 0.005
epochs = 1500
loss_fcn = F.mse_loss

# Set up optimizer to optimize the model's parameters using Adam with the selected learning rate
opt = optim.Adam(params = model.parameters(), lr=lr)

# Training loop
for epoch in range(epochs):
    out = model(x) # Evaluate the model
    loss = loss_fcn(out,y) # Calculate the loss -- error between network prediction and y

    loss_curve.append(loss.item())

    # Print loss progress info 25 times during training
    if epoch % int(epochs / 25) == 0:
        print(f"Epoch {epoch} of {epochs}... \tAverage loss: {loss.item()}")

    # Move the model parameters 1 step closer to their optima:
    opt.zero_grad()
```

```
    loss.backward()
    opt.step()
```

```
Epoch 0 of 1500...       Average loss: 0.4744042158126831
Epoch 60 of 1500...      Average loss: 0.09833034127950668
Epoch 120 of 1500...     Average loss: 0.022363871335983276
Epoch 180 of 1500...     Average loss: 0.002071716357022524
Epoch 240 of 1500...     Average loss: 0.0014897839864715934
Epoch 300 of 1500...     Average loss: 0.0011637693969532847
Epoch 360 of 1500...     Average loss: 0.001006523729301989
Epoch 420 of 1500...     Average loss: 0.0008361965301446617
Epoch 480 of 1500...     Average loss: 0.0006972100818529725
Epoch 540 of 1500...     Average loss: 0.0005636118003167212
Epoch 600 of 1500...     Average loss: 0.0004662715655285865
Epoch 660 of 1500...     Average loss: 0.00039241608465090394
Epoch 720 of 1500...     Average loss: 0.00032601619022898376
Epoch 780 of 1500...     Average loss: 0.0002884758869186044
Epoch 840 of 1500...     Average loss: 0.00026762307970784605
Epoch 900 of 1500...     Average loss: 0.0002573389501776546
Epoch 960 of 1500...     Average loss: 0.00025288251345045865
Epoch 1020 of 1500...    Average loss: 0.000250950368354097
Epoch 1080 of 1500...    Average loss: 0.00025012611877173185
Epoch 1140 of 1500...    Average loss: 0.00024947928613983095
Epoch 1200 of 1500...    Average loss: 0.00024902066797949374
Epoch 1260 of 1500...    Average loss: 0.00024853990180417895
Epoch 1320 of 1500...    Average loss: 0.00024804612621665
Epoch 1380 of 1500...    Average loss: 0.00024762339307926595
Epoch 1440 of 1500...    Average loss: 0.0002470871841069311
```

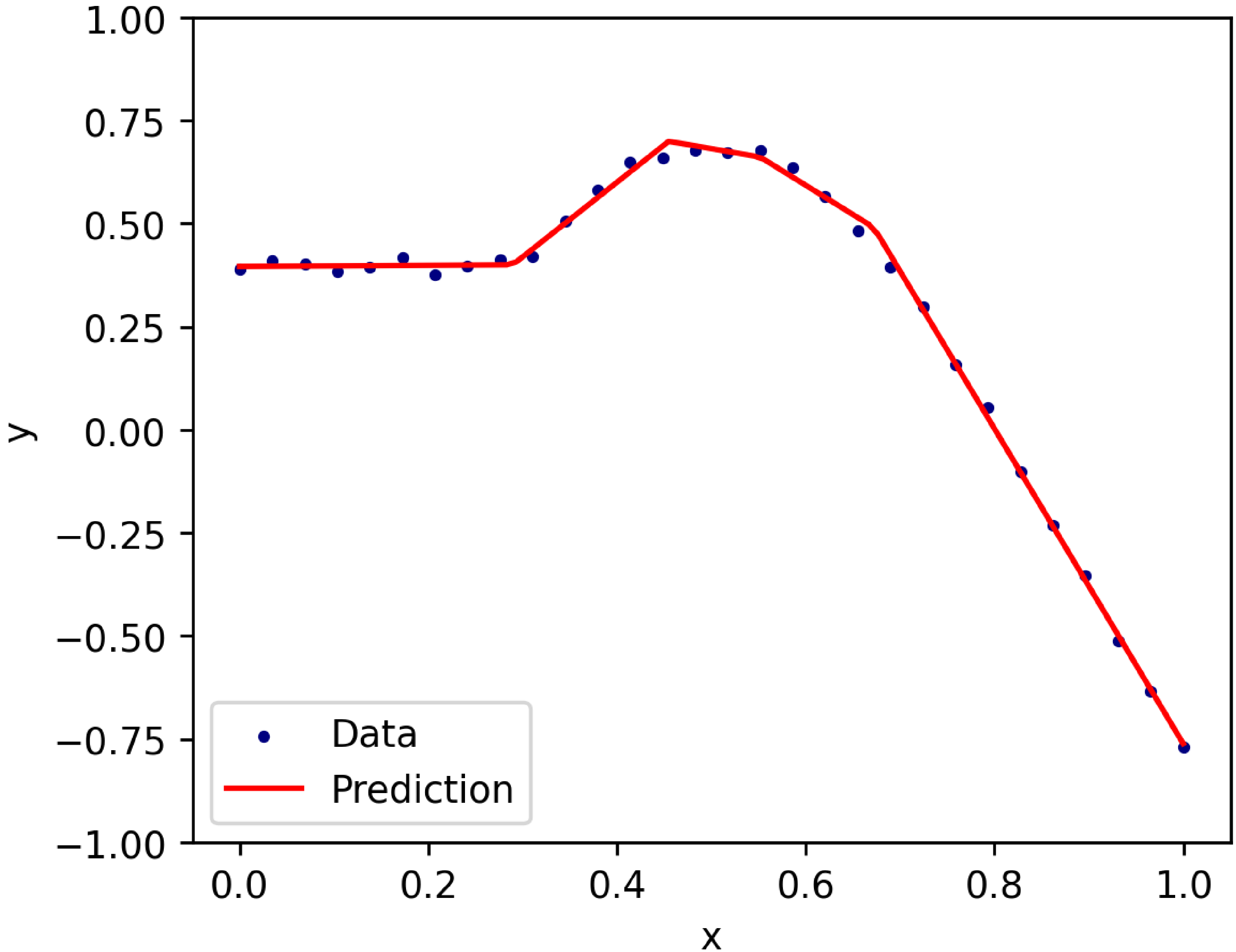
In [39]:

```
plt.figure(dpi=250)
plt.plot(loss_curve)
plt.xlabel('Epoch')
plt.ylabel('Loss (MSE)')
plt.title('Loss Curve')
plt.show()
```



In [40]:
```
xs = torch.linspace(0,1,100).reshape(-1,1)
ys = model(xs)
```

```
plt.figure(figsize=(5,4),dpi=250)
plt.scatter(x,y,s=5,c="navy",label="Data")
plt.plot(xs.detach().numpy(), ys.detach().numpy(),"r-",label="Prediction")
plt.legend(loc="lower left")
plt.ylim(-1,1)
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
y
0.0
0.2
0.4
0.6
0.8
1.0
x
Data
Prediction

Your Turn

In the cells below, create a new instance of `Net_2_layer`. This time, use 20 neurons per hidden layer, and an activation of `F.tanh`. Plot the loss curve and a visualization of the prediction with the data.

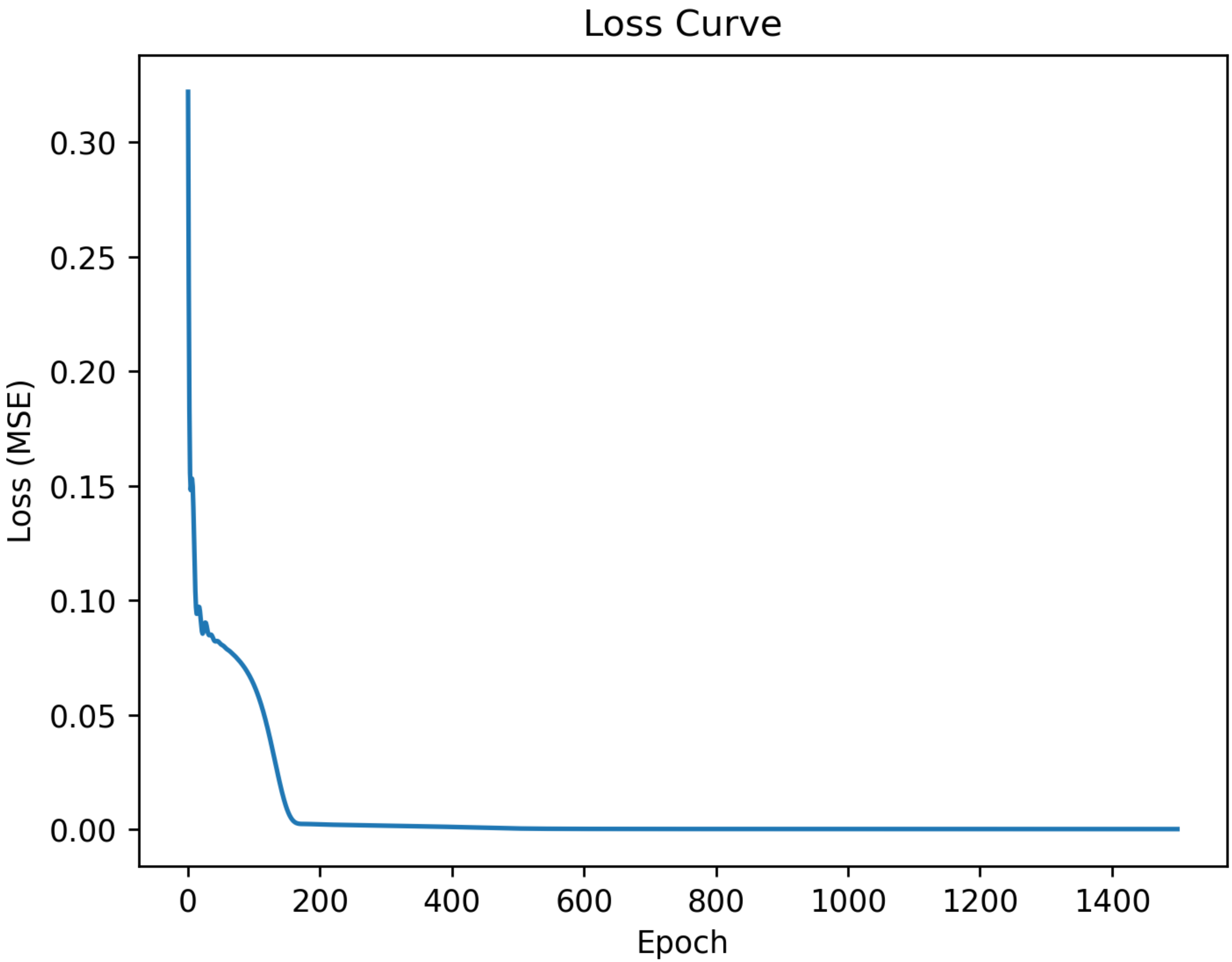
In [41]:
```
# YOUR CODE GOES HERE
model = Net_2_layer(N_hidden = 20, activation = F.tanh)
loss_curve = []
```

In [42]:
```
lr = 0.005
epochs = 1500
loss_fcn = F.mse_loss

opt = optim.Adam(params = model.parameters(), lr=lr)

# Training Loop
for epoch in range(epochs):
    out = model(x)
    loss = loss_fcn(out,y)

    loss_curve.append(loss.item())

    if epoch % int(epochs / 25) == 0:
        print(f"Epoch {epoch} of {epochs}... \tAverage loss: {loss.item()}")

    opt.zero_grad()
    loss.backward()
    opt.step()
```

```
Epoch 0 of 1500...      Average loss: 0.3219418227672577
Epoch 60 of 1500...     Average loss: 0.07849700003862381
Epoch 120 of 1500...    Average loss: 0.044798869639635086
Epoch 180 of 1500...    Average loss: 0.0024474714882671833
Epoch 240 of 1500...    Average loss: 0.002022543689236045
Epoch 300 of 1500...    Average loss: 0.0017344001680612564
Epoch 360 of 1500...    Average loss: 0.0014220451703295112
Epoch 420 of 1500...    Average loss: 0.0010031410492956638
Epoch 480 of 1500...    Average loss: 0.0005750214913859963
Epoch 540 of 1500...    Average loss: 0.0003725489950738847
Epoch 600 of 1500...    Average loss: 0.00030775790219195187
Epoch 660 of 1500...    Average loss: 0.0002830736921168864
Epoch 720 of 1500...    Average loss: 0.00027152951224707067
Epoch 780 of 1500...    Average loss: 0.00026513481861911714
Epoch 840 of 1500...    Average loss: 0.00026069581508636475
Epoch 900 of 1500...    Average loss: 0.0002569577773101628
Epoch 960 of 1500...    Average loss: 0.0002534904342610389
Epoch 1020 of 1500...   Average loss: 0.0002501792914699763
Epoch 1080 of 1500...   Average loss: 0.000247005868004635
Epoch 1140 of 1500...   Average loss: 0.00024397413653787225
Epoch 1200 of 1500...   Average loss: 0.00024108236539177597
Epoch 1260 of 1500...   Average loss: 0.00023832437000237405
Epoch 1320 of 1500...   Average loss: 0.00023568874166812748
Epoch 1380 of 1500...   Average loss: 0.00023316477017942816
Epoch 1440 of 1500...   Average loss: 0.0002307366085005924
```

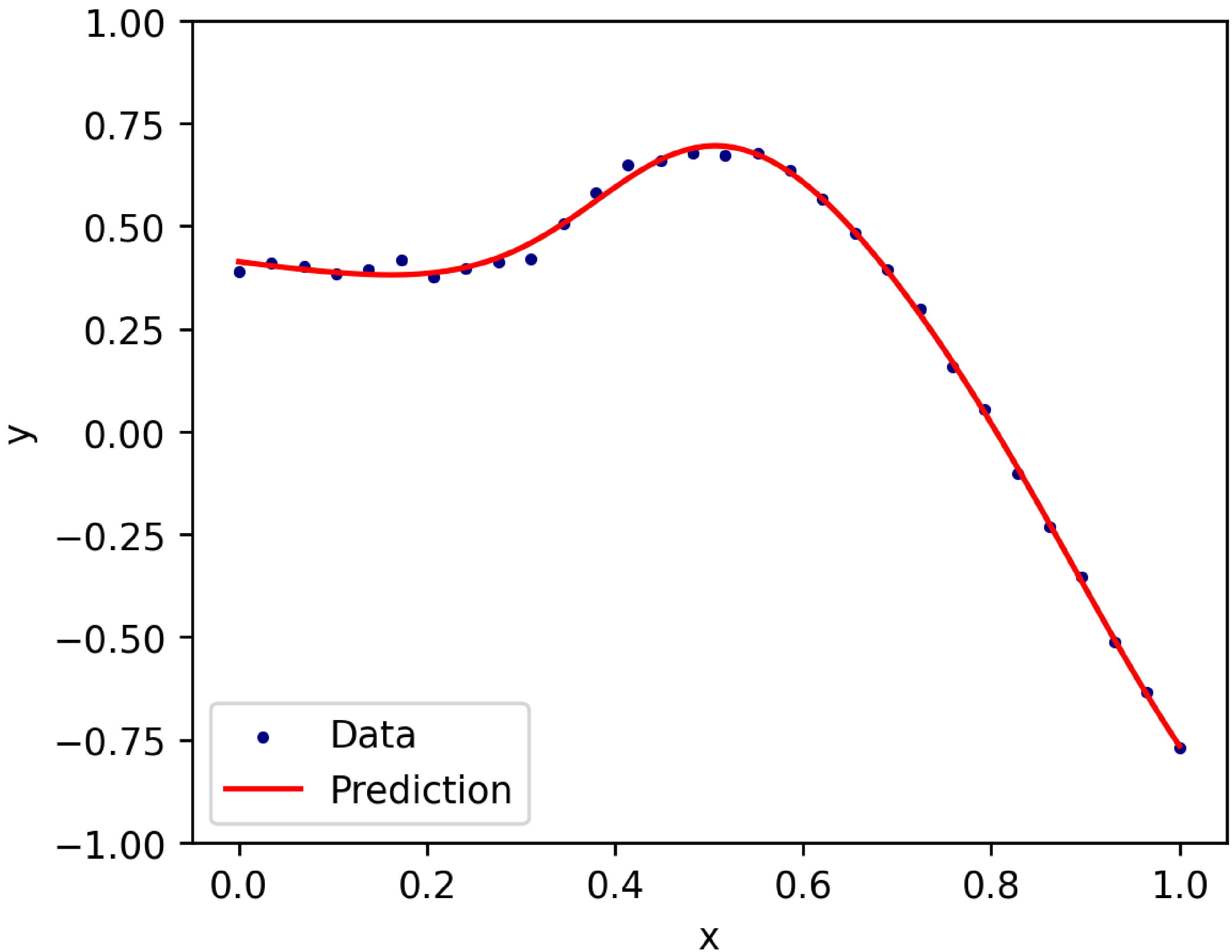
In [43]:

```
plt.figure(dpi=250)
plt.plot(loss_curve)
plt.xlabel('Epoch')
plt.ylabel('Loss (MSE)')
plt.title('Loss Curve')
plt.show()
```



In [44]:
```
xs = torch.linspace(0,1,100).reshape(-1,1)
ys = model(xs)
```

```
plt.figure(figsize=(5,4),dpi=250)
plt.scatter(x,y,s=5,c="navy",label="Data")
plt.plot(xs.detach().numpy(), ys.detach().numpy(),"r-",label="Prediction")
plt.legend(loc="lower left")
plt.ylim(-1,1)
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
y
0.0
0.2
0.4
0.6
0.8
1.0
x
Data
Prediction

```
In [ ]:
```